Coordinated Group Movement Between Polygons

1. Introduction

In the context of modern computer games, the navigation and the behaviour of friendly NPCs is as important for the immersion as the behaviour of hostie NPCs. In many stealth games, friendly NPCs are trying to hide with the player, moving from one "shadow" area to another, while staying hidden. The problem of moving tens of groups at the same time, while maintaining the formations and creating a realistic looking behaviour is a non-trivial task. It is also important to point out that the complexity of the problem increases with a non-trivial geometry of the map. In this report, we will discuss our approach on a group movement from one polygon to another.

2. Required Knowledge

In order to understand the problem and the method presented in this paper, one must have some understanding of *Pathfinding*, *Group Movement* and *Steering Behaviours*.

2.1 Pathfinding

Pathfinding is the core of any AI movement system whose responsibility is finding a path, if one exists, from any point of the game map to another. A pathfinder usually uses some sort of a precomputed data structure to store the result path and might use it to guide the movement. There are a lot of existing algorithms for achieving this, the algorithm used in this paper is A* algorithm. Given the start and the destination node, it employs best-first search algorithm to find the least-cost path. The path cost between nodes is calculated by the distance and the heuristic function.

2.2 Group Movement

The pathfinding algorithms discussed in the previous section are not well suited for a group movement, which assumes a movement of two or more agents as a single unit. Thus, group movement algorithms are built on top of the known pathfinding algorithms. One of such algorithms is David Silver's *Cooperative Pathfinding* algorithm [2020]. This is a reservation-based algorithm, which adds one more dimension to the existing dimension space. The additional dimension is the time dimension, which helps to predict the future positions of the moving agents.

2.3 Steering Behaviours

A lot of complexity for simulating game movement arises from the required data of the surroundings and the future path that must be calculated in advance. Steering behaviours [2012] is one of the approaches trying to eliminate this complexity. It aims to aid AI controlled agents move in a realistic manner by using improvisational navigation around the agent's environment. Such a simple method can produce rather complex movement patterns like "Flee and Arrival", "Pursuit and Evade", "Queue", "Path Following" and others. In method described in this paper the only used movement patterns are: "Avoidance" and "Seeking Patterns", which help the groups pursuit they destination and dynamically avoid different types of obstacles on their path.

3. Method

Before diving into the solution presented in this paper, it is important to familiarize yourself with the initial setup of the environment for the proposed solution. The things that require extra attention are the Map and the Groups.

3.1 Map

The map is the medium for the movement of the AI agents in the simulated environment. The map presented in this solution is a grid-based map comprised of columns and rows of "Nodes".

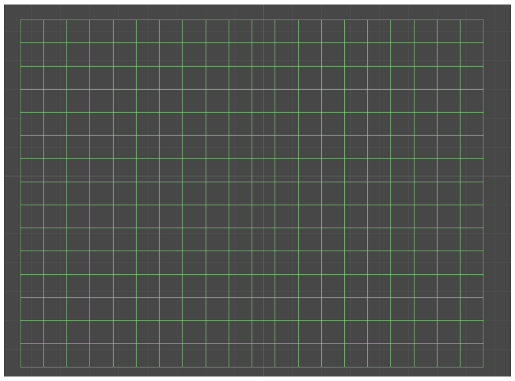

Figure 1. *The grid-based Map.*

The number of nodes differ from map to map. However, the size of each node is assumed to be the same for the scope of the project and are assumed to fit each group within the cell. Note that larger sizes of nodes reduce the time for path calculation while decreasing the accuracy of the "walkable" space of the map.

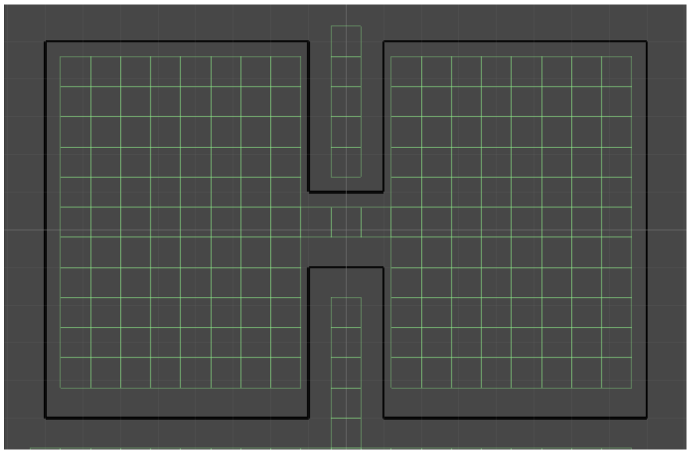

Figure 2. Larger node grid with less "walkable" space accuracy and better performance

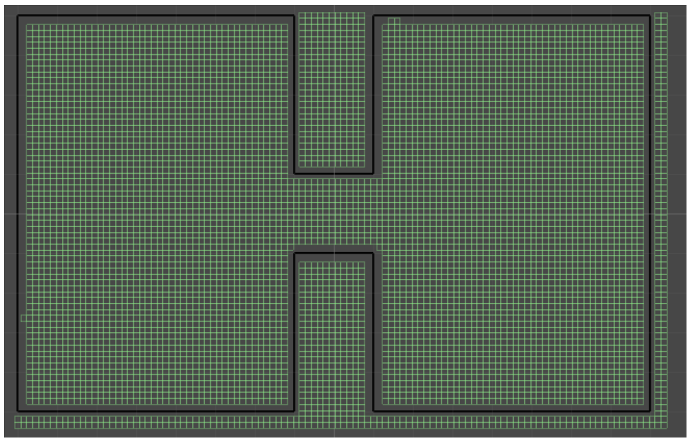

Figure 3. *Smaller node grid with better "walkable" space accuracy and worse performance*

3.2 Groups

A group is comprised of individual moving agents. The following assumptions are made for the purpose of this project.

- Groups contains exactly 6 agents
- All agents move at the same speed

It is also important to introduce the following properties of the group that will be used in the proposed solution to the problem.

- Each group has a center.

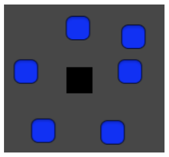

Figure 4. *The black square represents the group's centroid*

- Each agent "remembers" the initial relative position to the group center
- Each group has a dynamic bounding rectangle, which is calculated as group moves

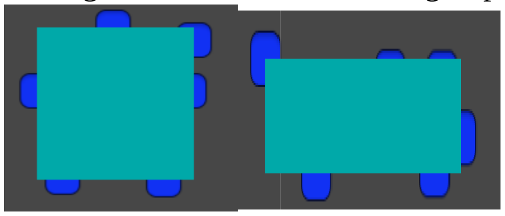

Figure 5. *The blue rectangle represents the bounds of the group for different formations.*

- There is a limit on the distance an agent can travel from the center of the group.

3.3 Movement

One of the core systems is the movement system. There are 2 ways that will be presented in this report:

- The naïve approach

  As the name suggests, this is the simplest approach to the movement in the given setup. This approach uses the smaller node grid and the original A* algorithm to calculate the path for each agent of the group. The resulting path is adjusted by adjusting the destination using the save relative position from the center of the group. This kind of movement requires a grid with a significant number of nodes which substantially slows the performance.

- The main approach

  In order to improve the performance and make the solution applicable to the large number of groups a grid with larger and smaller number of nodes was employed. A* is still used, however, only once from the centroid to the destination. Once the path is determined, the members of the group follow it using steering forces, what results in more realistic and smoother movement patterns.

Now I will describe the algorithm **PS 1** used and presented in **Pseudocode** section. First, on line 2, we calculate the path from the group center to the destination, using A* algorithm. Next, on lines 4-6 each agent is notified about the path. Once a path is assigned to an agent, it will follow it by applying steering forces. Note that the final destination of each agent is adjusted according to relative positions to the centroid.

3.4 Group Coordination

When trying to move multiple groups around the map multiple problems arise that must be addressed.

1. The destination can already be occupied by some other group.
2. Even if the destination node is available some other groups can intersect the path to the destination.
3. When moving number of groups from one part of the map to another, where exactly does each group go?

To overcome these problems all the groups must be notified about the movement of each other. Every new movement must be coordinated with other groups to be made possible. Before the group's path to the destination is calculated, we need to check if the node is available. If it is not available, the group at the destination must be moved to the nearest available node. However, what if the node is reserved by some other group on its way to its destination?

In this case we need to adjust our destination to the nearest node. This simple trick solves the 3rd point as well. When moving number of groups from one area to another, a center of a new area can be chosen as the new destination, so that once the destination node is reserved by one of the groups other groups adjust accordingly and reserve the neighbouring available nodes.

In **PS 2** of **Pseudocode** section, on lines 7 – 13, we check if there is a stationary group at the intended destination. If there is, on line 9, we get the available neighbour of the node using a breadth-first search starting from the destination node. Once the available destination is found, we move the stationary agent there, on line 11, to free the destination for incoming group. However, if the destination is not occupied, we still need to make sure that the destination does not lie on a path of any other group. On lines 16-23, we iterate through all the other groups and if any other group is already travelling there, we again find the available neighbour, but this time set it as a destination to the current group.

3.5 Group Scheduling

Once we have the solution for multiple group movement a new problem arises. Imagine the following setup:

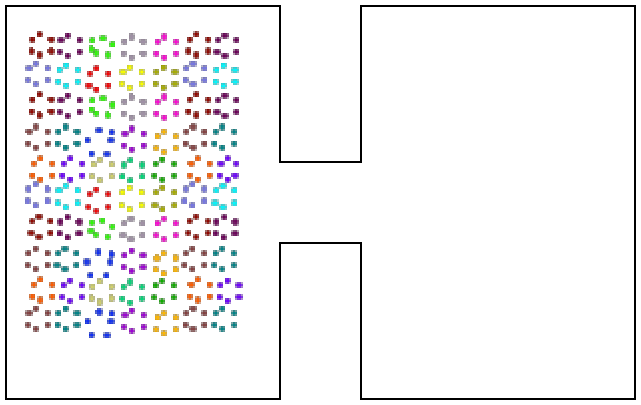

Figure 6. *"H-shaped" map with than 70 groups.*

The small, coloured rectangles are the agents of each group. Groups are comprised of the agents of the same colour. Consider that the intention is to move all the groups from the left side of the "H-shaped" map to the right side through the narrow junction in the middle. If all the groups move at the same time to their respective destinations all of them will rush through the junction creating a very unrealistic movement pattern and a huge number of collisions. Thus, a scheduling algorithm must be introduced.

Given the direction of movement for each agent, it is possible to create the movement collections of groups with different priorities to create the "layer-by-layer" type of movement. First, all the groups ray cast in the direction of their individual movement, from 3 points: group centroid and 2 diagonal points of the bounding box within a certain distance passed as a parameter. After that step, each group is aware of any other group in front and are assigned in the movement groups. Once the so-called movement groups, a group consisting of groups, are formed the movement is initiated according to the distance from the destination. It is important to note, that the right to reserve the closest destination is given to the groups the furthest from it to minimize the number of intersections.

To give a more in-depth explanation of the algorithm, I will describe **PS-3** algorithm of **Pseudocode** section. The first step, on line 3, is to sort the groups by the distance to a destination. Since we need the furthest group, we reverse the list on line 4. On lines 6-9, the path to the destination is precalculated for each group, to "reserve" their destinations. Moreover, the movement is disabled, so that the agents do not start moving before their turn is scheduled. Then on lines 10-24, the raycasting is in the direction of their path is performed. Once we get the "hit" result, the groups are sorted to the layers or "movement groups" to be moved sequentially. Note that the direction is not past to the raycast call, because the call on line 9 sets the path to the respective agent. The logic for creating the "movement groups" is simple, if there are no raycast hit results it means that there are no group obstacles in front, thus the groups are to be first to move. Else if there are hit results, that means that there are other groups in the front of us in the direction of movement. Thus, we find the index of the "movement group" in front and assign the next to the current one. The next step is to enable the movement of the groups. However, before moving the groups, we need to calculate the waiting time, on line 27 the time is defined as one over the distance between the centroid of the "movement groups". Then the *baseWaitTime* passed as an argument is used to find the end time by multiplying it on line 32 and then capped by the *maxWaitTime* .

4. Map Design

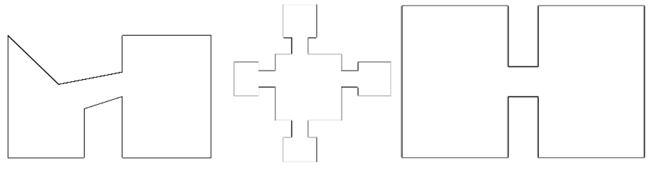

***Figure 7**. 3 maps used to test the approach. "Diagonal", "Four Corners", "H-shaped".*

The algorithm presented was tested on the 3 types of maps. They all have a similar layout with one or multiple "choke points". The layout of these maps makes the movement and scheduling of groups from one area to another through the junction a non-trivial task. All the maps were tested with 500 agents, what made around 40 independent groups clustered together in an area of the map.

5. Metrics

One of the main metrics used in the project was the time for all the groups to identify their destinations and calculate the paths. The movement process itself was not timed as it is heavily dependent on the implementation of the movement and the initial variables such as speed. Besides measuring the times, the algorithm took to complete on different iterations and maps a different algorithm must be used for comparison. For this purpose, the naïve algorithm, described before, was used.

5.1 Results

The algorithm was tested on 3 different scenarios. All the scenarios include the "H-shaped" map.

- Scenario 1 – 4 groups, naïve approach.
- Scenario 2 – same as Scenario 1, but with approach described in the paper
- Scenario 3 – same approach as in Scenario 2, however, with 70 groups.

Table 1 presents the results of 10 successive iterations of different scenarios.

Scenario 1	Scenario 2	Scenario 3
1.690s	0.008s	0.571s
1.699s	0.007s	0.554s
1.70s	0.008s	0.559s
1.714s	0.008s	0.574s
1.702s	0.008s	0.578s
1.768s	0.008s	0.559s
1.686s	0.008s	0.564s
1.714s	0.008s	0.566s
1.699s	0.008s	0.572s
1.762s	0.008s	0.564s

Average:

1.713s	0.008s	0.566s

***Table 1**. Results of 3 scenarios*

6. Conclusion

The results presented in section **5.1** demonstrate a significant improvement in performance. Given the exact same conditions, the approach presented, on average, is 214 times faster than the naïve approach. From the results, it can be concluded that the algorithm presented is suitable for movement of many groups in different types of maps, including complicated geometries.

6.1 Improvements

Besides code optimizations, there are different approaches that can be taken in order to improve the performance.

1. Heatmaps

   Once we move one group out of the way of the other group, we need to find the next destination. The use of heatmaps, which are calculated dynamically as each group crosses a boundary of a grid node, can improve the performance as well as result in a better-looking movement pattern.

2. Groups being entities

   Instead of having independent agents grouped together, creating one entity responsible for all the updates.

3. Using Navigation Mesh
   This approach might show increased performance over the approach presented here, however, it would require a completely different reasoning, since the base assumptions will be completely different.

References

1. David Silver (2020), *Cooperative Pathfinding*, Retrieved from https://www.davidsilver.uk/wp-content/uploads/2020/03/coop-path-AIWisdom.pdf
2. Fernando Bevilacqua (19 October, 2012), *Understanding Steering Behaviours*, Retrieved from https://gamedevelopment.tutsplus.com/series/understanding-steering-behaviors--gamedev-12732?fbclid=IwAR1pe7bZ41QEO95by-kYIjx4DbGqLLuCM-jqlxHwg3QylWD4utY1q1Mb7zM

Pseudocode

```
function MoveGroup(group, destination){
   path = FindPath(group.Center, destination);

   foreach(agent in group){
       agent.FollowPath(path);
   }
}
```

PS 1. *Algorithm for group movement.*

```
function MoveGroupToPoint(group, destination)
{
   occupied = false;

   foreach(otherGroup in groups){
       if(otherGroup == group) continue;
       if(otherGroup.Path.IsEmpty && otherGroup.GridPosition == destination){
           occupied = true;
           otherGroupDestination = destination.GetAvailableNeighbour();
             MoveGroupToPoint(otherGroup, otherGroupDestination);
            break;
        }
    }
        if(!occupied){
        foreach(otherGroup in groups){
            if(otherGroup == group) continue;
            if(!otherGroup.Path.IsEmpty && otherGroup.Path.Last == destination){
                destination = destination.GetAvailableNeighbour();
                break;
            }
        }
    }
  MoveGroup(group, destination);
}
```

PS 2*. Algorithm for group coordination.*

```
function MoveGroups(groups, destination, baseWaitTime, maxWaitTime){

  groups.SortByDistanceTo(destination);
  reversedGroups = groups.Reverse();
  movementGroups = new    List<List<Group>>();
   foreach(group in reversedGroups){
       group.DisableMovement();
       MoveGroupToPoint(group, destination);
   }
    foreach(group in groups){
        raycastResult = group.Raycast();
        if(raycastResult.IsEmpty){
            if(movementGroups.IsEmpty){
                movementGroups.Add(new List<Group>());
            }
            movementGroups[0].Add(group);
        }
        else{
            index = get index of the list containing the raycast result.
            if(movementGroups.Count <= index){
                movementGroups.Add(new List<Group>());
            }                   movementGroups[index].Add(group);
        }
    }
    for(i = 0; i < movementGroups.Length; i++){
        if(i == 0)waitTime = 0;
        else waitTime = 1 / DistanceBetweenCentroids(movementGroups[i], movementGroups[i - 1]);
     }

        foreach(group in movementGroups[i]){
            movementGroups.EnableMovement();
        }
        waitTime = baseWaitTime * waitTime;
        if(waitTime > maxWaitTime) waitTime = maxWaitTime;
        Wait(waitTime);
  }

}
```

***PS 3** The algorithm for group scheduling*